

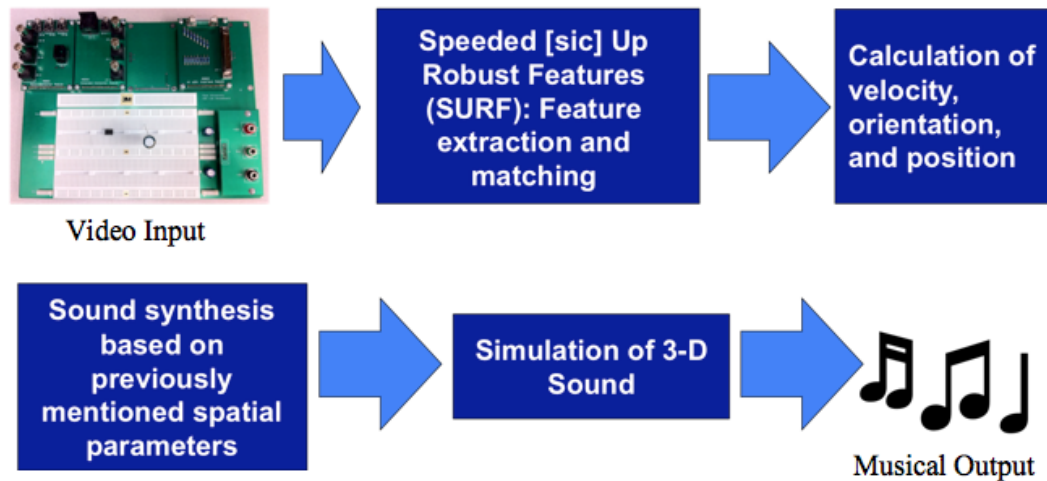
1. [Introduction](#)
2. [Background](#)
3. [Calculating the Position, Speed, and Orientation of an Object](#)
4. [Object Detection Challenges and Solutions](#)
5. [Musical Mapping](#)
6. [3-D Sound Implementation](#)
7. [Results](#)
8. [Conclusions](#)
9. [Future Work](#)
10. [Poster](#)
11. [References](#)
12. [The Team](#)

Introduction

This is the introduction to our group project: DWTS - Dancing With Three-Dimensional Sound.

Since the introduction of the American Disabilities Act some years ago, our society has made commendable strides towards ensuring that people with mental and physical handicaps can navigate our world with the same ease as everyone else. However, a significant amount of work remains to be done, especially in the entertainment and performing arts industries. For example, consider a mother who wishes to attend her daughter's ballet recital, but cannot see the performance due to acute blindness; such a situation struck us as tragically unfair, and became the motivation for our signal-processing project.

Once these sentiments coalesced into a driving force for our team, we set out to discern the best strategy with which to aid the blind in their endeavor to experience performances of dancing. Two potential courses of action immediately presented themselves: either we could utilize signal-processing to directly counteract blindness in the eyes, or we could take advantage of one of the other innate human senses to communicate the phenomenon of dancing in a nontraditional, yet intuitive way. Since none of us were particularly well versed in the science of ocular anatomy, we opted for the latter option, and managed to implement a system that captured the intricacies of dancing by transforming a visual input into three-dimensional sound using a predetermined musical mapping. The general methodology that we adhered to in realizing our system is as follows:



Object Detection and 3-D Sound Synthesis Procedure

The rest of this module will provide an in-depth illustration of these facets of our project.

Background

This module contains the background of our object detection algorithm and 3-D sound synthesis system.

Object Detection

Finding an object in an image can be done in various ways. One of the most common methods used is matching certain features in the image to those in the object that is being searched for. Traditionally, these features have included mostly edges and corners because these are relatively easy to find, but unfortunately these features are also often sensitive to even small amounts of noise and are also often neither scale nor rotation invariant. Another method of object detection in an image is color matching. This method works reliably and efficiently, but is very sensitive to things in the background and thus, in general, a very plain background must be used in order to track the desired object correctly. For this reason, we found that a good tradeoff between speed of computation and sensitivity to noise was the Speeded Up Robust Features (SURF) algorithm. Using this algorithm allows the program to track an object by simply providing a picture of the desired object, and it works both outdoors and indoors without any overly tight constraints on the environment.

Procedure

The SURF algorithm is actually based on the earlier Scale Invariant Feature Transform (SIFT) algorithm, but has a few modifications that make the former a lot faster to compute. In order to use these algorithms, one first simply takes a picture of the object that is to be tracked; this image is called the "template." Then, the algorithm finds "keypoints" on another image; these are places with interesting features that might be useful for tracking an object, but not necessarily the object we're looking for. Each of these keypoints includes a scale, orientation, coordinates, and other identifying information ("descriptors"). In order to actually find where the object we're looking for is, one finds the keypoints whose descriptors are closest to those calculated for the template image. This is usually done by finding the keypoints in the test image with the least squares difference in their descriptor values as compared to those in the template descriptors.

SIFT will be described first because it is simpler, and then the differences between SIFT and SURF will be noted.

Creating the Scale-Space

The first step in the SIFT algorithm is to create what is known as a "scale-space." This basically creates a set of images that consist of versions of the original image, blurred to different extents. This is important because oftentimes there are objects of very different scales in an image. An example of this would be a large tree with lots of small leaves on it. Adding higher amounts of blur means that only larger objects will remain detectable while small amounts of blur will preserve small details. Additionally, this step is critical for maintaining scale invariance so that the object can be detected when it is farther or closer to the camera in the video than when the template image was taken. Blurring of the image is done by taking the 2-D convolution of the image with a Gaussian function. This has the same effect as passing the image through a low pass filter in the frequency domain. It is important to note that most current methods of generating scale-spaces use the Gaussian and not other types of low pass filters to perform this blurring because it has been mathematically proven that the Gaussian filter does not add any features to the source image that were not originally there. Other types of filters might potentially add things like edges in places where there were originally none.

The convolution in question is:

The Gaussian function is:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

Gaussian Function

Note that in $G(x, y, \sigma)$, x and y actually represent the distance of the current pixel from the origin in the x and y directions, respectively. For $I(x, y)$, x and y are simply the coordinates of the current pixel. As σ increases, the amount of blur and thus the type of scale that is preserved increases. Some example Matlab code is as follows:

```
Source = im2double(rgb2gray(imresize(imread('MainImage.jpg'),  
[480,640])));
```

```
Variance = 100;
```

```
for x=1:480
```

```
for y=1:640
```

```
R(x,y) = (1/sqrt(2*pi*Variance))*exp((- (abs(x-240)^2+abs(y-  
320)^2))/(2*Variance));
```

```
end
```

```
end
```

```
Blurred = conv2(Source,R)/(2*sqrt(Variance));
```

```
imshow(Blurred)
```





The convolution of the image with the Gaussian will not be the same size as the original image, so cropping is necessary to get something like the bottom image. Additionally, convolving by the Gaussian function in 2-D is separable, which means that using a convolution matrix to get the 2-D convolution gives the same result as applying a 1-D convolution in both directions. This is useful because it makes computation much, much faster, but it is also a bit more involved. Matlab takes care of all these issues through the `fspecial` and `imfilter` functions. For example, the previous code can be replaced with the following:

```
Source = im2double(rgb2gray(imresize(imread('MainImage.jpg'),  
[480,640])));
```

```
Variance = 100;
```

```
h = fspecial('gaussian',[480,640],sqrt(Variance));
```

```
imfilter(Source,h); %convolves source with h and performs necessary  
cropping
```

```
imshow(ans);
```

Later Steps

SIFT operates on grayscale images because for each pixel only one byte of information is required (intensity) so the convolutions are a lot faster. It is also worth mentioning that the size of the Gaussian filter used doesn't need

to be the size of the entire image, but larger filters result in stronger blur. In SIFT, the original image is downsampled by increasing factors of 2, and each of these images is then blurred to different levels. The number of downsampled images is called the number of "octaves", and can be chosen by the programmer. Normally 4 octaves are used with 5 levels of blur each for a total of 20 images.

The next step involves taking the difference of these blurred images. In each octave, each image has its neighbor with a higher level of blur subtracted from itself except the last one since there is nothing after it. The effect of this step is leaving an image where edges, corners, and other features have higher contrast than places that are less well defined.

Example:

```
h1 = fspecial('gaussian',[8,8],1.5);
```

```
h2 = fspecial('gaussian',[8,8],10);
```

```
LowBlur = imfilter(Source,h1);
```

```
HighBlur = imfilter(Source,h2);
```

```
imshow(HighBlur-LowBlur);
```



Difference of Gaussians

From these Difference of Gaussian (DoG) images, minima and maxima are located; these are the initial sets of keypoints. Next, a contrast check is done that throws away keypoints that are not above some threshold contrast with respect to their neighboring pixels; this helps increase the accuracy of the algorithm. Lastly, for each keypoint, descriptors including scale and orientation are calculated by using neighboring pixels.

An important difference between SIFT and SURF is that SURF does not downsample images by factors of 2 and instead just keeps a "stack" of images at the same resolution, but with different levels of blur. Then, instead of finding the difference of Gaussians, SURF approximates second-order derivatives by using boxcar filters; this results in a much higher calculation speed, but with sometimes lower rates of detection accuracy.

Fortunately, there are many implementations of SURF available in various programming languages. In initial testing during this project, OpenCV's C++ implementation of the algorithm was used, and it was almost fast enough to track in real time using a 640x480 resolution webcam input. Since a lot of the sound code we already had was done in Matlab, however, we instead opted for Dirk-Jan Kroon's Matlab adaptation of OpenSURF, originally a very fast C++ implementation by Chris Evans.

Example Results

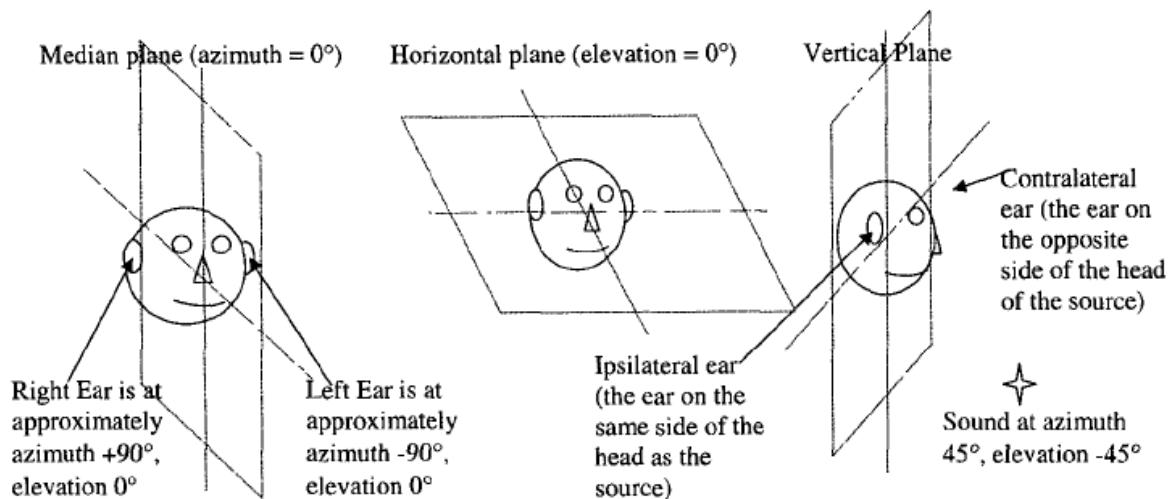
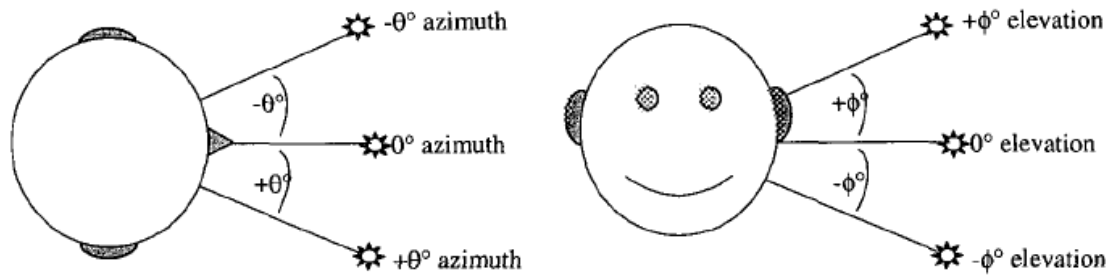


Matched keypoints: Scene on left,
Template on right

The 3-D Sound Effect

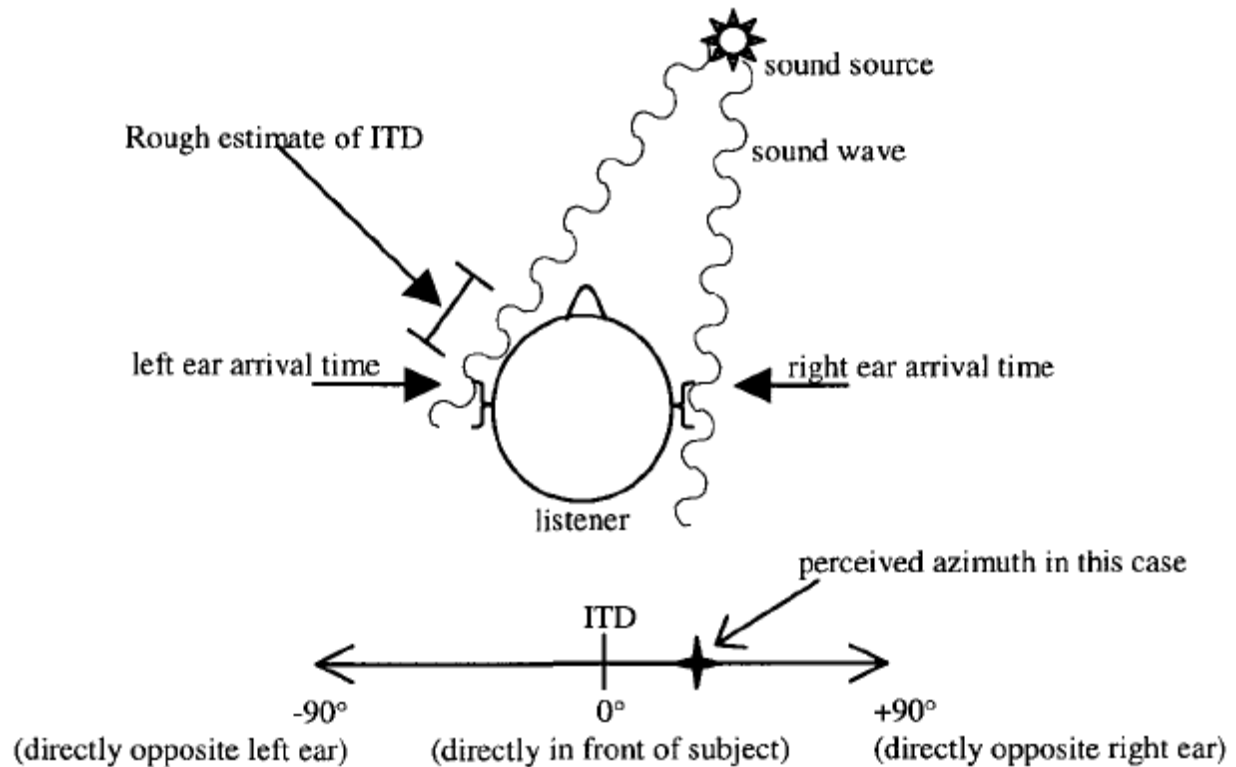
Definition

The 3-D sound effect allows the listener to locate the exact position of an object in space. The position is described in terms of an azimuth and elevation. The following figure illustrates the concepts of azimuth and elevation, as well as the spatial coordinate system we used in our project.



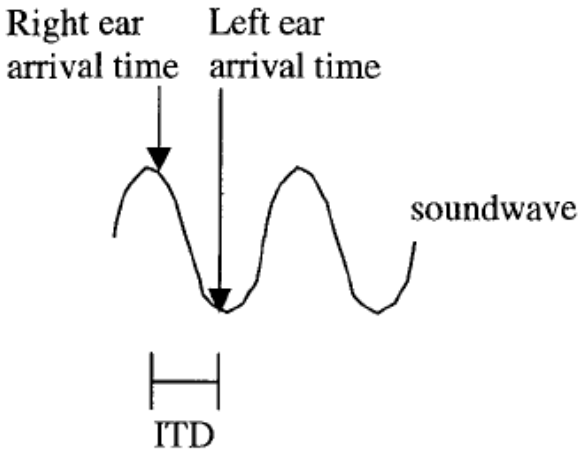
The Spatial Coordinate System Implemented in Our Project

One of the most common ways to synthesize 3-D sound is to use the head-related transfer function, which includes information about the interaural time difference (ITD), interaural intensity difference, and spectral coloration of a sound made by the torso, head, outer ears, or pinna. The interaural time difference is defined to be the difference in time between when the wavefront arrives at the left ear and when the wavefront arrives at the right ear (Figure 7). The source of sound is perceived to be closer to the ear at which the first wavefront arrives. As such, a larger ITD corresponds to a larger lateral displacement.



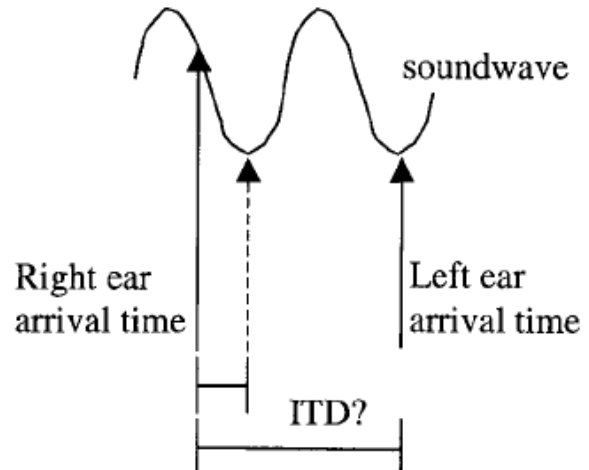
Interaural Time Difference

Using ITD, we can decide the azimuth of the sound source. Unfortunately, when the frequency is above 1500 Hz, the wavelength of the sound is approximately the diameter of the head. Therefore, different azimuths might have the same ITD, and aliasing will occur (Figure 8).



In this example, the right ear receives the signal first, so the sound will be perceived to the right of the listener. Since the ITD is less than a wavelength, the ITD represents a phase difference which corresponds to a unique azimuth

(a)

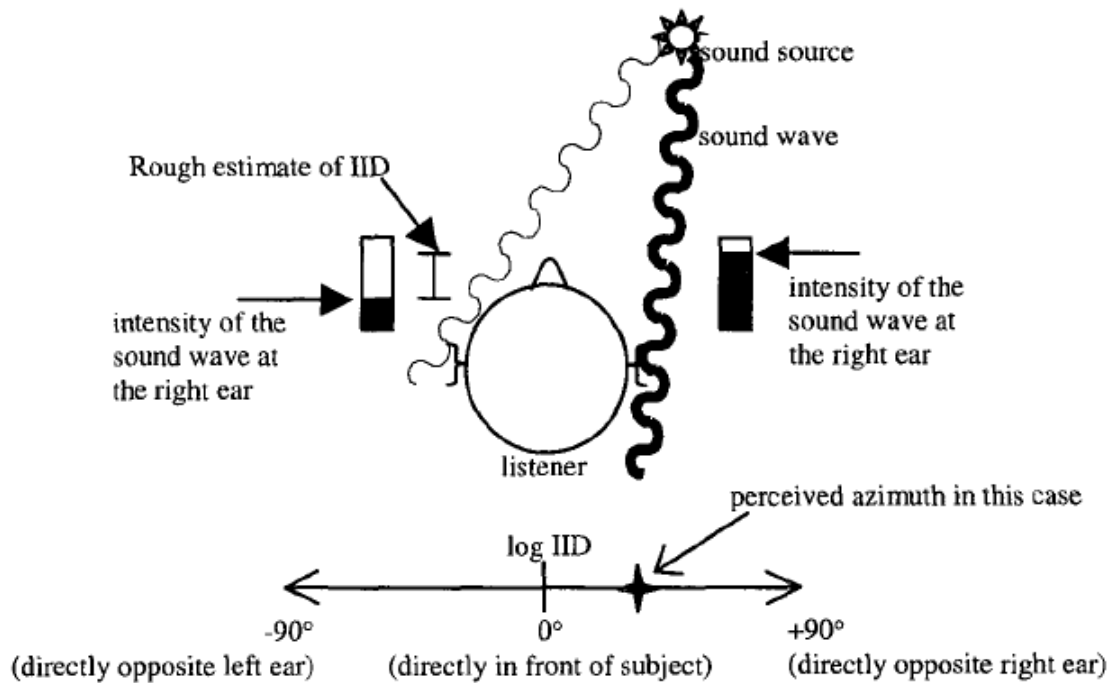


Here, the ITD is longer than a wavelength. In this case, the ITD does not correspond to a unique azimuth, as the auditory system may perceive a shorter ITD due to aliasing problems.

(b)

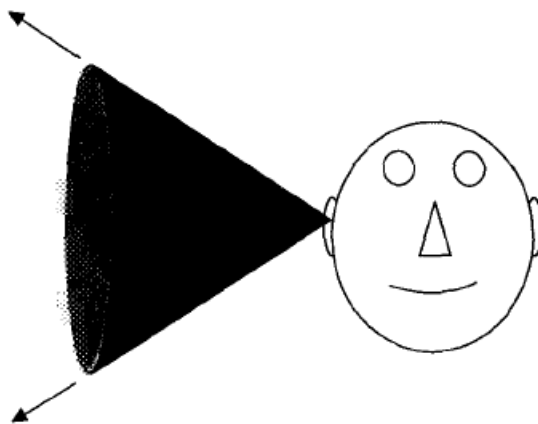
Ambiguity of ITDs in determining the lateral position for high frequencies: a) Below 1500 Hz and b) Above 1500 Hz

In order to uniquely determine the lateral position of the sound source, the interaural intensity difference (IID) is introduced to the HRTF. The interaural intensity difference is defined as the amplitude difference between the wavefront that arrived at the left ear and the wavefront that arrived at the right ear (Figure 9).



Interaural Intensity Difference

Utilizing both ITD and IID, we can find the azimuth of the sound source, but not its position in space. In fact, there are infinitely many points in space which have the same ITD and IID. Those points form a so-called “cone of confusion” (Figure 10).



Cone of Confusion: All points on the cone

at the same distance from the cone's apex
share the same ITD and IID.

In order to distinguish sounds originating from points on a cone of confusion, the HRTF we used in our project includes the spectral cues produced by the structure of the ears. With the ITD, IID, and these spectral cues, our system can exactly locate the position of sound source in space.

Calculating the Position, Speed, and Orientation of an Object

This module describes how to calculate the position, speed, and orientation of an object.

Thanks to the many resources available on the Internet, we were able to utilize Dirk-Jan Kroon's Matlab adaptation of OpenSURF as the core function of the image processing part of our project.

OpenSurf returns a set of parameters for all the features above the set threshold. These parameters include x-position (scalar), y-position (scalar), scale (scalar), orientation (scalar), Laplacian (binary), and descriptor (vector). These parameters are put into a structure that is associated with each feature. Hence, if two hundred features are found to be above the threshold restriction, there will be two hundred structures in the output, each with cells such as x, y, and etc. The descriptors, however, are essential to feature matching between two images.

Matching Features

With the help of the example code also written by Kroon, we understood how to find matches between all of the features found and returned by the function OpenSurf.

If two images, I1 and I2, are processed by OpenSurf (I1 is the test image, while I2 is the template), two structures, T1 and T2, are returned. In order to match the features, the descriptor for each feature in the template needs to be compared to the descriptor for each feature in the test image (or the other way around). To do so, descriptors need to be extracted and put into a matrix for convenient usage. Kroon's way of doing this is by making each descriptor a column vector and augmenting all such vectors into a matrix with dimensions "length of descriptor \times number of features". After doing this we will have two matrices, D1 and D2. To compare two vectors in multidimensional space, one would naturally think about least-squares-error, and this is indeed how the matching is done.

To avoid a nested for loop and increase computational speed, one column vector from D1 is taken at a time and is replicated (by tiling) into a matrix

of the size of D2; then the least squares can be done in a matrix. Entry-wise differences are calculated between the newly constructed matrix and D2, and then the entry-wise square is taken. Each column is then summed and a distance error vector is generated. Next, we take the minimum distance and record the feature number from both I1 and I2 of the associated distance. After all the found features for D1 are compared with those of D2, a nice minimum distance vector is generated. From here it is quite easy to rank the similarity of the matches based on the minimum distance.

Of course not all matched features will be used. On one hand it is sufficient to have just a couple of matched points to find the same object. On the other hand, not all found features above the threshold are good matches. They might just be distinct features generated by OpenSurf, but might end up not matching anything on our target at all; in fact, they might just be a similar contour in a complex background. We call these features “astray points” because later when we try to pin down the center of the object, they introduce lots of error. The number of points that are used is set to be a variable so as to introduce some flexibility.

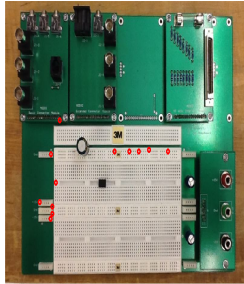
After selecting the eight best matches (the number we chose), we can use their feature number to access their individual information, such as position, in each picture.

Once we figure out how to find the matches and extract their relevant information, we need to calculate the data for our sound generation module. In a dance, it is the relative position of the body parts, their moving speed, and orientation that convey the most information; as a result, our first goal is to find out the position of the object’s center, how fast the center of the object is moving, if it is skewed, and how fast it is rotating.

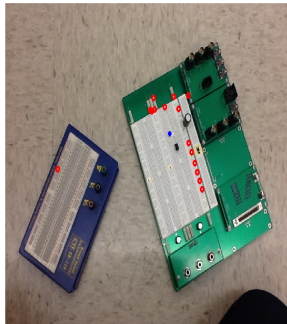
Position and Orientation

We took an easy approach to calculate the center of the tracked object. The center is calculated by averaging the x and y indices with all the selected feature points. One might argue that this method is not robust against error. However, because of the special properties of the images we used (video, which means frames are blurry; distant objects, which lower the resolution

even more), we can hardly tell if a point is an “astray point” or if the object is very close to the camera.

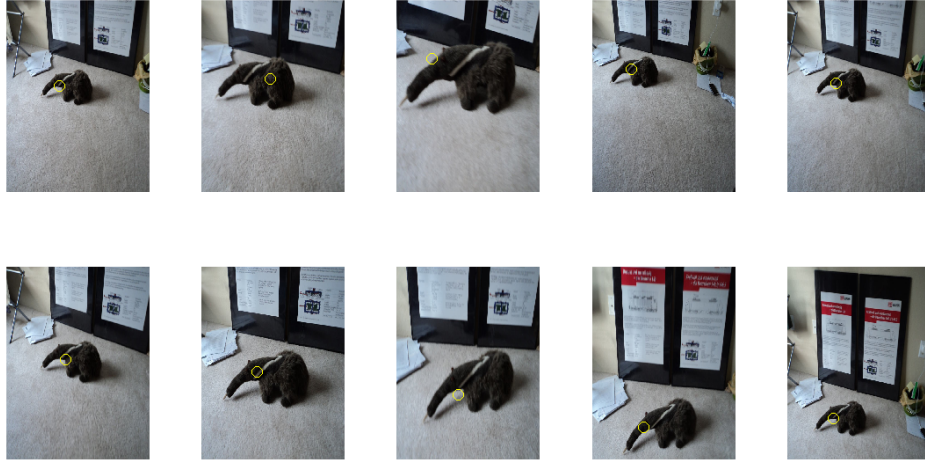


(a)



(b)

Figure (1) is the template, Figure (2) is the test image. The blue star is the calculated position of the target object. Notice that if the number of selected points is large enough, one astray point would not affect the result that much.



A series of test images are run using the same template. Central position of the target object (the anteater) is found.

Orientation is not extracted from the feature descriptors; rather, it is calculated using matrix transformation (affine matrix).

Put the x and y positions from I1 into a column vector and augment all of these column vectors from different features into a matrix and call it X1. We can do the same thing for I2 and call it X2. Then we have

where A is the transformation matrix that contains information such as x and y scale and rotation angle. Thus,

$$A = \begin{bmatrix} S_x \cos \alpha & S_y \sin \alpha \\ -S_x \sin \alpha & S_y \cos \alpha \end{bmatrix}$$

where the S's are scaling factors. To find out the angle, we can simply do X1/X2 and then divide the entry (1,2) by (2,2) of the resulting matrix, and it gives $\tan(\alpha)$. We can then take the inverse tangent to find out the angle.

Video Processing and Speed Calculation

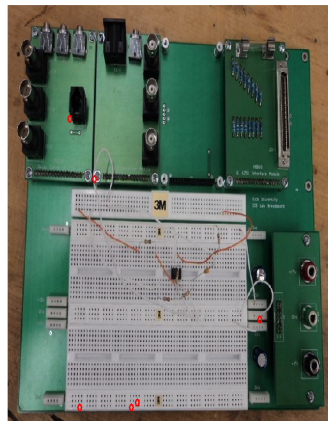
First, the video is recorded and completely loaded into MATLAB. We use one picture for the template; then we compare the template against each single frame using OpenSurf, and the position and orientation for the target object in that frame is calculated in the main for loop; this takes a while to run. Once the process is done, speed can be calculated. To ensure the same length of all output vectors, the first entries for translational velocity and rotational velocity are both set to 0. Later entries are calculated by the difference of position or angle (then divided by 30 because there are 30 frames per second) between two frames. The units for translational velocity are somewhat arbitrary because in this project we are not tracking the radial distance, and hence absolute speed cannot be calculated in a metric unit. The units for translational velocity are pixels per second; angular velocity is in radians per second.

Object Detection Challenges and Solutions

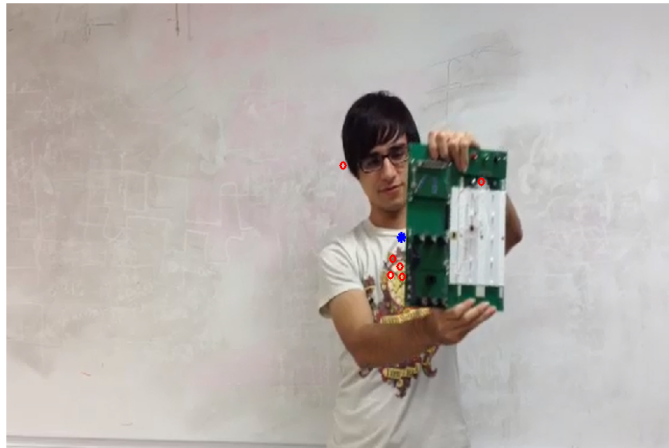
This module is about the challenges we faced when detecting objects and the solutions to them.

Resolution Difference Between Template and Video

In the beginning, we made the template image and the video separately. In our experiment we used the large green breadboard as the object to be tracked. We first took a picture of the breadboard lying on the table using a high-resolution camera. Then, we switched the camera to video mode with a size of only 480×640 and recorded a video with someone moving the board around. We ran the code and the tracking result was disappointing.

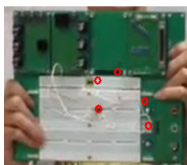


Template with high resolution

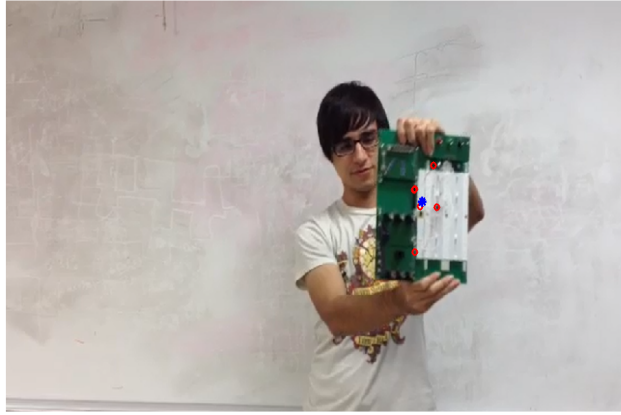


Tracked result – off target (the “matched” points do not even make sense)

Then we realized that lots of details in the template image could not even be seen on the object in the video. For example, the screws and pinholes were not at all visible. Hence, we decided to crop out the template from a screenshot of the video so as to ensure the same level of detail. And it worked!



Template that is
cropped out
from the same
source video



Tracking result of a video frame

Motion Blur and Video with Lower Resolution

Another problem with matching video frames is that when the shutter speed is relatively slow, the image in each frame blurs and the features of the object are no longer distinguishable.



Motion

-

Blurred
Image



Motion-Blurred Image



Clear
image



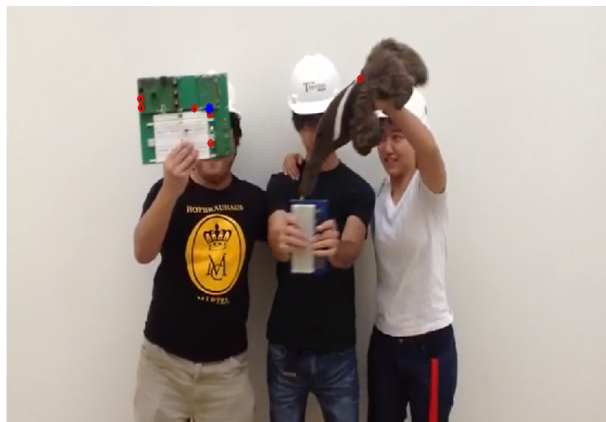
Clear image

To avoid motion blur, we had to slow down our movements when recording the video. Otherwise, the tracked position would be completely off.

However, the overall resolution and quality of the video matter, too. We tried converting one of our video files online because one member's MATLAB cannot read in a .mov file. But after converting the file, the quality of the tracking result went down dramatically. We believe that it was partially due to the even lower resolution of the converted file. When the resolution is too low, a sharp corner can no longer be distinguished from a round corner. On the other hand, there could also be a problem introduced by the compression method that was used by that website, because clearly we could see non-uniformity in the once uniform white board area. This compression method actually introduced additional features or errors into the video frames.



Before
video
conversion
n



Before video conversion



After

video
conversion
(also
notice that
the ratio
of the
image is
changed)



After video conversion (also notice that
the ratio of the image is changed)

Complex Background

Since the resolution of our recording devices was not very good, a complex background could also be problematic, because there would be plenty of similar features in the background, but there would not be enough details to

fully distinguish them. As a result, we had to use a simple background, such as a wall, and had to avoid wearing shirts full of complex patterns.



Features of
template
are
inaccuratel
y matched
to similar
features in
the
background



Features of template are inaccurately
matched to similar features in the
background

Objects Without Distinct Features

An object that doesn't have distinct SURF features is extremely problematic. We were experimenting with a blue pencil box. If the pencil box, lying on the table by itself, was used as the template image, OpenSurf would not output any feature descriptor that was above the threshold. Not until we lowered the threshold all the way down to 0.00001 (it was always set to 0.0008 for our project) did we find six features (normally more than a dozen for a small template and more than a hundred for a full video frame), and even those six features were degenerated!



Only six
repeated
features
are
found
when
threshold
is set
to
0.00001

If we cropped the template in such a way that some background was included, then only the outline or features in the background would be detected, and looking for matching features in the testing image would not make any sense. As is shown below, the matching failed.



All
tracking
points
are
attracted
to
irrelevant
edges.



All tracking points are attracted to irrelevant edges.

In order to avoid all of the problems that occurred during our experiments, we decided that for the demo video we would only have one dancer, so as to get a closer shot and a higher resolution. The person would wear plain clothing, act in front of a wall, and move slowly and avoid out-of-plane rotation in order to increase the accuracy of object tracking.

However, because the feature matching is not very robust, and even one or two mismatches would introduce huge error into the orientation of the object, the output angular velocity is extremely noisy. That is why we finally decided to discard the angular velocity data.

Musical Mapping

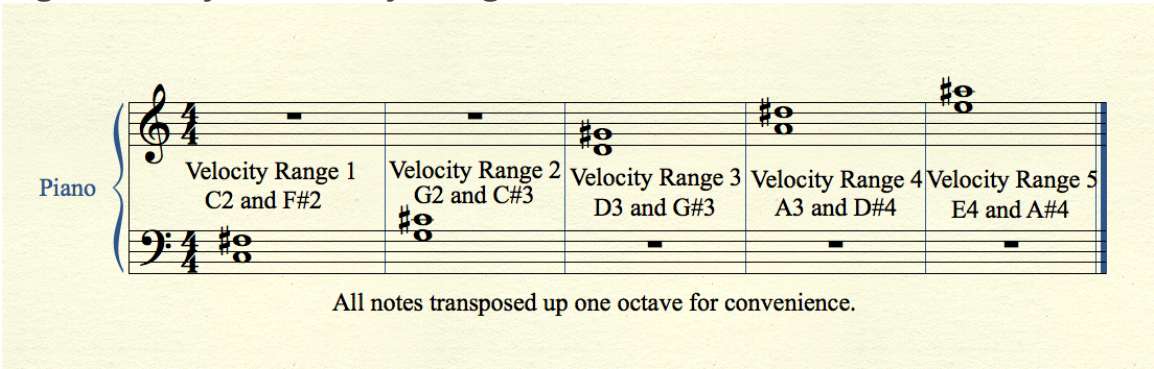
This module describes how we do musical mapping in our project.

The musical mapping that formed the backbone of our system focused on three main components: the type of object in question, its velocity, and its orientation. Each of the three distinct objects that were held by the dancer as she moved corresponded to a different musical instrument, as outlined in the table below.

Table 1: Mapping Objects to Instruments	
Object	Instrument
Large, Green Breadboard	Guitar
Small, Blue Breadboard	Horn
Plush Anteater	Oboe

Once each object was linked to the appropriate instrument, we needed to construct a relationship between the velocity of an object at any given time and a distinct pair of notes. Due to the fact that it is unfeasible to have a different pair of notes for every possible velocity, we opted to assign pairs of notes to certain velocity ranges, as seen in Figure 1, below.

Figure 1: Object Velocity Ranges



One essential feature of these pairs of notes is that the interval between the two notes for any given velocity range forms what is known in music theory as a tri-tone. This means that while either of the two notes in a certain velocity range played alone will sound comfortable to the listener, when they are combined into a chord, the resultant sound will be characterized by an extraordinary amount of dissonance. Indeed, the dissonance that is created is so pronounced that a tri-tone has historically been known as the

“Diablo de Musica”. While such an explicit clashing effect may appear to be a significant setback to the overall performance of our system, we found that it can actually be extremely effective as a representation of an object’s orientation. Thus, our system (as can be seen in the code attached later in this module) delineates the orientation of the three objects used by the dancer according to the methodology specified in Table 2.

Table 2: Musical Mapping of an Object’s Orientation	
Object Orientation	Notes Heard In Sound
Vertical	Note 1 only
Horizontal	Note 2 only
At an angle between vertical and horizontal	A weighted mixture of Note 1 and Note 2

Accordingly, if an object is placed in a vertical position, only one of the two notes in the appropriate velocity range will be played by our system. Similarly, if an object is placed in a horizontal position, only the other note will be heard in the resultant sound. However, if an object is placed in a position between vertical and horizontal (i.e. at an angle), a mixture of the two notes in the appropriate velocity range will be played, with more weight placed on the note nearest the object’s actual orientation. Thus, if an object is positioned nearly vertically, the note corresponding to the vertical position will predominate; if an object is oriented nearly horizontally, the opposite note will be emphasized more heavily. If an object is oriented at a perfect forty-five degree angle, each of the notes will be played with equal emphasis.

3-D Sound Implementation

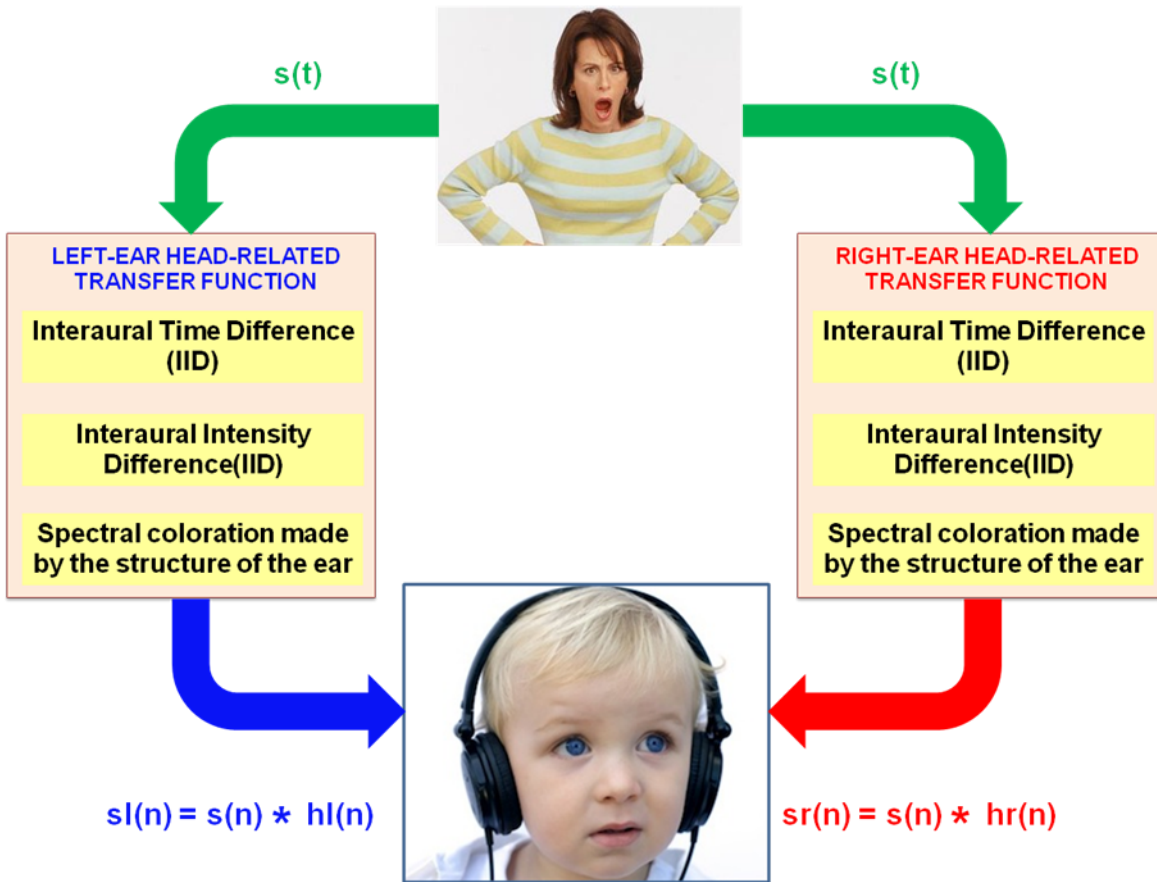
This module demonstrates how our 3-D sound system was implemented.

We employed the head-related transfer functions (HRTFs) measured by [Bill Gardner](#) and Keith Martin at the MIT Media Lab to implement the 3-D sound effect. Gardner and Martin determined the left and right ear impulse response from a Realistic Optimus Pro7 loudspeaker mounted 1.4 meters from a KEMAR dummy head microphone at a sampling rate of 44.1 KHz.

Additionally, we used the compact data file package published on their website, which included a data-reduced set of 128 point symmetrical HRTFs derived from the left-ear responses. Because of symmetry, the right-ear responses can be derived from the left-ear responses. Measurements were made at elevations from -40 degrees to +90 degrees and at azimuth from 0 to 180. More detail on HRTFs measurements can be found at:

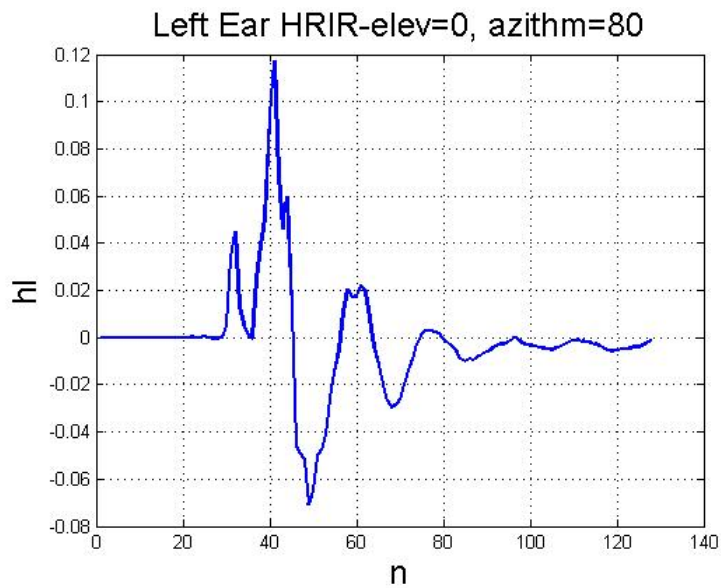
<http://sound.media.mit.edu/resources/KEMAR/hrtfdoc.txt>

Utilizing Gardner and Martin's HRTFs, we built the following system in Matlab to synthesize a 3-D sound.

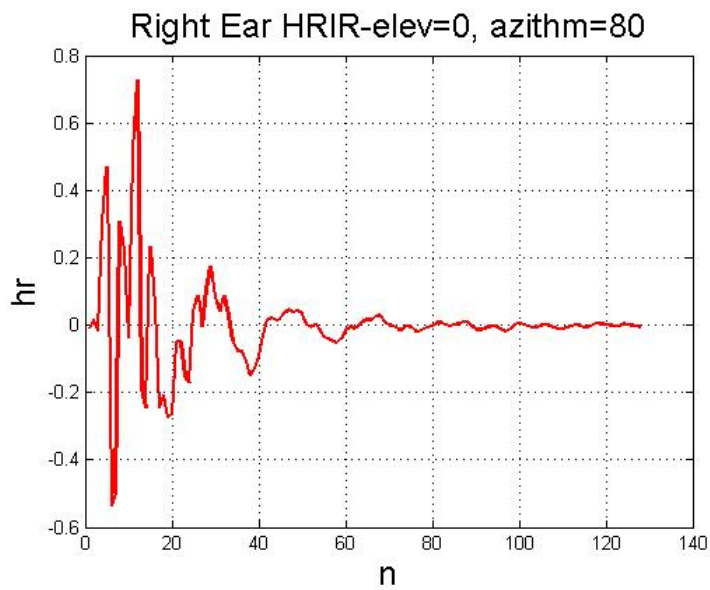


3-D Sound Synthesis System

At an elevation of 0 degrees and an azimuth of 80 degrees, the impulse responses of the two systems above are shown in figure 2 and 3.



Left-Ear Impulse Responses of 3-D Sound
Synthesis System at elev = 0 and azimuth =
80



Right-Ear Impulse Responses of 3-D Sound
Synthesis System at elev = 0 and azimuth =
80

For those who love mathematics, Corey I. Cheng and Gregory H. Wakefield provided a complete mathematical derivation of head-related transfer functions in their paper *Introduction to Head-Related Transfer Functions(HRTFs): Representations of HRTFs in Time, Frequency, and Space*.

Results

This module includes the results of our project.

We successfully tracked the circuit boards and the anteater when the video image was clear, which thereby enabled us to track the movement of the dancer's head and hands. However, the requirement that the video image must be clear put many restrictions on the possible dance moves allowed.

After obtaining the positions of the objects, as well as their orientations and velocities from the OpenSurf function, we used musical mapping and 3-D sound effects to create an audio file that demonstrated the movements of Zhiting's hands and head while she was dancing as represented by the movements of the two circuit boards and the anteater. We combined this audio file with the movie we made using Windows Movie Maker. We let a graduate student listen to the movie file we made while closing his eyes in our poster presentation; he then told us how the hands were moved. His answer was close to the real movements of the dancer's hands. However, the movement of the left hand was not tracked very well, so instead of being heard from the left side of the head, the horn sound was heard from the right side of the head most of the time. Furthermore, the similarity between the horn sound and the oboe sound made it difficult to distinguish between the movements of the left hand and the head.

Conclusions

This module includes the conclusions of our project.

Conclusion

The OpenSurf function can be used to detect not only an object, but also its motion. As a result, by utilizing the OpenSurf function, we are able to track the dancing movements of a dancer who is wearing a hat and holding objects in her/his hands. As a result, the 3-D sound generated from the output of the OpenSurf function successfully describes those movements. Though work remains, our object detection and 3-D sound synthesis system can be used to help the blind experience the TV show Dancing With The Stars, as well as many other dancing performances.

Future Work

This module contains our future work.

An important aspect of enjoying dancing is listening to the music with which the dancers are dancing. Incorporating this music into the sound synthesized by our project would be a great addition. Another improvement would be to increase the accuracy of object detection, because the current method of finding the object's center relies on averaging, and this is susceptible to outliers. The greatest improvement would be to do away with the requirement of holding an object while dancing and being able to track the dancer's hands and feet themselves so that a wide array of dance videos would be eligible for processing in our system. Lastly, we need to find a way to implement the Doppler effect on the final sound without distorting it.

Poster
This is the poster we made for our project.



RICE
Unconventional Wisdom

Zhitong Cai, Emmanuel Herrera, Minh Nguyen, Reed Shoho
Department of Electrical and Computer Engineering



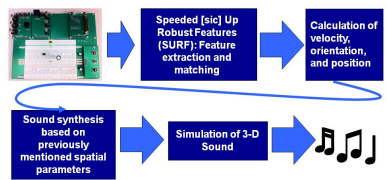
RICE ece
redefining limits

DWTS: Dancing With Three-Dimensional Sound

Motivation

Objective: Make a ubiquitous art form, dancing, more readily available to individuals with restricted vision; to accomplish this goal, we will engineer a system that facilitates the conversion of the actions of dancing into three-dimensional sound.

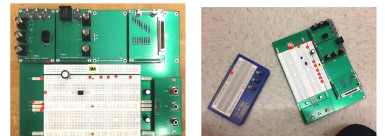
Procedure



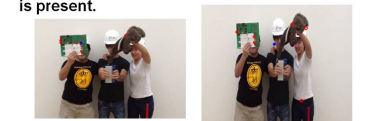
Object Feature Detection

- Feature Detection was done using the SURF Algorithm.
- Step 1: Blur the image to different degrees.
 - Step 2: The difference between images with varying amounts of blur is calculated; only edges and corners are preserved.
 - Step 3: The image is divided into 3x3 grids, and the central pixel of each of these grids is compared to its neighbors.
 - Step 4: The brightest and darkest points are found, and for each of these, a magnitude, gradient, and orientation is calculated.

Object Tracking Results

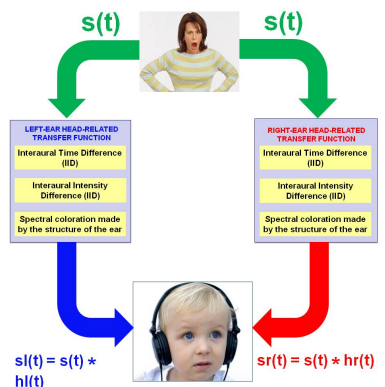


Successful tracking of features on the green board even though a similar object (blue board) is present.

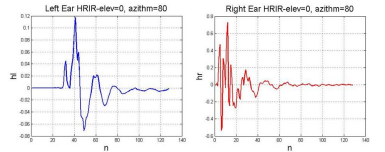


When the image blurs, SURF confuses similar features on irrelevant objects with those that we are looking for.

3-D Sound Synthesis System

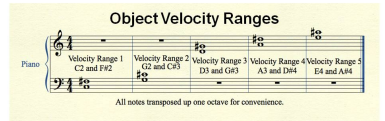


Head-Related Impulse Response



Musical Mapping

Object	Instrument
Large, Green Circuit Board	Guitar
Small, Blue Circuit Board	Horn
Plush Ant eater	Oboe



Future Work

1. Discard points that introduce large error in the position calculation.
2. Track the radial movement of the object.
3. Make the sound continuous and fluid.

Acknowledgements

We would like to thank our mentor, Eva Dyer, along with Dr. Richard Baraniuk, Dr. Don H. Johnson, and Mr. Michael Dye for their guidance and assistance.

A list of references is available on a separate handout.

Our Poster

References

This module contains the references we used in our project.

1. <http://sound.media.mit.edu/resources/KEMAR.html>
2. <http://www.mathworks.com/matlabcentral/fileexchange/28300-opensurf-including-image-warp>
3. <http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/>
4. http://www.cs.cf.ac.uk/Dave/CM0268/Lecture_Examples/
5. http://iie.fing.edu.uy/~rocamora/wind_synthesis/doc/
6. http://en.wikipedia.org/wiki/Head-related_transfer_function
7. <http://ses.library.usyd.edu.au/handle/2123/8218>
8. <http://www.jstor.org/stable/3680062>
9. <http://www.seventhstring.com/resources/notefrequencies.html>
10. <http://www-ece.rice.edu/~jdw/241/file.35.html>
11. <http://eyeborg.wix.com/neil-harbisson#!videoart>
12. <http://www.coranac.com/tonc/text/affine.htm>

The Team

This module is about our team and our contributions to the project.

This project was done by Zhiting Cai, Emmanuel Herrera, Reed Shoho, and Minh Nguyen. All of us are junior ECE students with an emphasis on signals and systems.

In this project, Minh Nguyen was responsible for implementing musical instruments in Matlab, synthesizing 3-D sound, and writing the function that mimics the Doppler effect to encode angular velocity into sound. He then combined all functions written by our group to code the Top_function and create the audio file for the final demo. Furthermore, he wrote the 3-D sound section of the poster. He also wrote the 3-D sound synthesis module and the conclusion module of the final report. Lastly, he collected the reports from other group members to publish them on Connexions.

Emmanuel Herrera carried out initial testing of a wide array of image detection methods including Haar Cascades, ORB and FREAK object descriptors, SIFT, and SURF. Additionally, he wrote the code to synthesize varying dissonance in the output sound depending on object orientation and did the work required to make this code compatible with output of the object tracking functions. Lastly, he wrote the Connexions module about "overview on SIFT/SURF," the text on our poster related to image tracking, and part of the future work.

In the earlier stages of this project, Zhiting Cai experimented with a bulb filter to do object detection. She then implemented the Matlab adaptation of the OpenSurf algorithm, calculated parameters for music module inputs, ran feature tracking experiments on videos, and devised the working conditions for the image processing module. She also worked on the content of the image processing and future work sections of our poster, and wrote the OpenSurf adaptation and problems and solutions for Connexions.

Reed Shoho was responsible for mapping the objects used by the dancer to musical instruments, as well as creating the various velocity ranges used by our system. The instantiation of each of these velocity ranges required him to implement a musical tri-tone utilizing a specific pair of musical notes. He was also responsible for the formatting and layout of the group's research

poster, in addition to composing the “Motivation” and “Musical Mapping” sections of that poster. Finally, he authored the introduction to the project’s Connexions module, in addition to writing the “Musical Mapping” section and serving as an editor for other group members’ writing to ensure greater clarity throughout the module.